В этом посте речь пойдет о функции scanf(), которая **выполняет форматированное чтение данных из стандартного входного потока stdin**. Эта функция имеет следующее определение (прототип):

```
int scanf(const char* format, ...);
```

Здесь первый параметр format – это указатель на форматную строку, на подобие той, что мы рассматривали в функции printf(). Последующее троеточие указывает на произвольное число дополнительных параметров, как правило, переменных. Функция возвращает целое значение типа int, равное числу прочитанных элементов из входного потока stdin.

Чтобы воспользоваться функцией scanf() нужно знать, как правильно задавать формат считываемых данных. Для этого, также как и в функции printf(), предусмотрены спецификаторы преобразований. Но они несколько иные.

Спецификатор	Описание
%d	Целое число со знаком в десятичной форме. (Приводится к типу int).
%i	Целое число в десятичной, шестнадцатеричной или восьмеричной системах. (Приводится к типу int).
%u	Целое беззнаковое (unsigned) число в десятичной форме. (Приводится к типу unsigned int).
%o	Целое число в восьмеричной форме. (Приводится к типу int).
%x, %X	Целое число в шестнадцатеричной форме. (Приводится к типу int).
%f, %e, %g %F, %E, %G	Вещественное число. (Приводится к типу float).
%c	Символ в соответствии с текущей кодовой таблицей. (Приводится к типу char).
%s	Строка (последовательность символов). Читается до первого пробела, перевода строки или символа табуляции.

Существуют и другие спецификаторы, но мы остановимся только на этих. Также спецификатор %s для ввода строк рассмотрим позже, когда будем проходить строки.

```cpp
#include <stdio.h>

int main(void)
{
    char byte;

    int count = scanf("%c", &byte);
    printf("count = %d, byte = %c\n", count, byte);

    return 0;
}
```

Давайте подробно разберемся, как это работает. Так как в формат-строке записан спецификатор "%c", то scanf() читает один байт из буфера входного потока stdin. Предположим, там находятся числа 100 и 53. Значит, функция читает первое значение 100. Далее, необходимо этот байт данных скопировать в переменную byte. И здесь возникает вопрос, как это сделать? Вначале, я напомню, что **любая переменная – это непрерывная последовательность байт**. В нашем примере – это одна ячейка, т.к. переменная byte имеет тип char. А значение переменной определяется тем, что записано в этих ячейках. То есть, для записи прочитанных данных из входного потока stdin в переменную byte достаточно в соответствующую ячейку памяти скопировать эти прочитанные данные. **Именно поэтому функции scanf() передается не значение переменной (как это было в функции printf()), а адрес переменной**. Забегая вперед отмечу, что оператор & перед именем переменной, как раз и возвращает адрес этой переменной. Зная этот адрес, функция scanf() имеет возможность менять значение переменной byte, записывая определенные данные напрямую в указанную ячейку памяти. Так происходит передача данных из входного потока в указанные переменные с помощью функции scanf().

Если данные были успешно прочитаны и занесены в переменную byte, то функция scanf() возвратит значение 1. Это говорит нам, что в одну переменную были успешно занесены данные из потока stdin.

Важно помнить, что во входном буфере вся введёная информация сохранятся. Так, если вы напишите:

```cpp
char a, b;
scanf("%c", &a);
scanf("%b", &b);
```

И введёте два идущих подряд символа, то они распределяться между этими функциями, а программа не будет ждать повторного ввода данных.

Пока, я думаю, все понятно. Давайте теперь поставим символ пробела между спецификаторами в форматной строке:

```cpp
int res = scanf("%c %c", &byte1, &byte2);
```

Этот пробел означает любые пробельные символы, которые могут присутствовать между двумя порциями данных. Сразу отмечу, что к пробельным относят символы: **пробела**, **перевода строки**, **табуляции** (и реже некоторые другие).

А теперь давайте вместо пробела поставим, например, запятую:

```c
int res = scanf("%c,%c", &byte1, &byte2);
```

При таком формате ввода будет ожидаться первый символ (любой), затем обязательно должна идти запятая, а затем еще один любой символ. Например, так:

c,d

А вот если входные данные не соответствуют формату, например:

cd

то функция scanf() **успешно прочитает только первый символ, а следующий (второй) оставит во входном потоке, т.к. вместо запятой записана буква d**. Переменная res в этом случае будет равна уже 1, а в переменной byte2 останется прежнее значение.

Конечно, мы можем комбинировать разные символы разделители в форматной строке, например, так:

```c
int res = scanf("%c, %c", &byte1, &byte2);
```

Тогда будет читаться первый символ, затем должна идти запятая, возможные пробельные символы, а затем, следующий не пробельный символ.

Чтение числовых значений из входного потока stdin

Я думаю, с чтением отдельных символов (байт) с помощью функции scanf() в целом все понятно. Теперь можно сделать следующий шаг и посмотреть, как выполняется чтение числовой информации из входного потока.

Если данные представлены в виде целых десятичных чисел со знаком, то для этого часто используют спецификатор %d. **Причем, этот спецификатор приводит целые числа к типу int**. И это очень важный момент. Сейчас я покажу почему.

```c
#include <stdio.h>

int main(){
        long long var_lli = 0;

        int res = scanf("%d", &var_lli);
        printf("res = %d: var_lli = %lld\n", res, var_lli);

}
```

Здесь всё круто до поры до времени. Например, если мы введём число 1234567890, то всё будет хорошо, но стоит его слегка расширить, например, до 12345678901234 и var_lli будет равно 1942892530. Всё из-за приведения спецификатором %d числа к типу int. Переменная могла бы хранить необходимое значение, да только нужно было разобраться с модификаторами. Вот таблица самых частых модификаторов:

Модификатор	Описание
h	%hd, %hi – для short int; %hx, %ho, %hu – для unsigned short
hh	%hhd – для signed char; %hhu – для unsigned char
l	%ld, %li – для long int; %lx, %lo, %lu – для unsigned long; %lf, %lg, %le – для double
L	%Lf, %Lg, %Le – для long double
ll (в стандарте C99)	%lld – для long long int; %llu – для unsigned long long

цифры	Максимальная ширина ввода (либо достигается максимальная ширина, либо служебный символ).
*	Пропуск данных.

Важно! Неправильно выбранный спецификатор может приводить к некорректному поведению программы. Даже если вы ставите модификатор пропуска данных (знак звёздочки), пожалуйста, следите за тем, чтобы дальше шёл правильный спецификатор, иначе последующие данные будут некорректными!!